Standard Code Library

FLself

SCUT

August 12, 2023

Contents

一切的开始

一些宏定义

- 需要 C++11

```cpp
// #define DEBUG
// #define InTerminal
// #define std_cpp17
#include<bits/stdc++.h>
#define int long long
#define PII std::pair<int, int>
#define VI std::vector<int>
#define VPII std::vector<std::pair<int, int> >
#define VVI std::vector<std::vector<int> >
#define ALL(a) (a).begin(), (a).end()
#define SIZ(a) ((int)(a).size())
#define FOR(i, l, r) for (int i = (l); i <= (r); ++i)
#define REP(i, r, l) for (int i = (r); i >= (l); --i)
#define lowbit(x) ((x) & (-(x)))
#define lbpos(x) (__builtin_ctz(x))
#define hbpos(x) (31 - __builtin_clz(x))

template<typename S, typename T> std::istream &operator>>(std::istream &is, std::pair<S, T> &pp) { is >> pp.first >> pp.second; return is; }
template<typename S, typename T> std::ostream &operator<<(std::ostream &os, std::pair<S, T> pp) { os << "(" << pp.first << ", " << pp.second << ")"; return os; }
template<typename S, std::size_t _siz> std::istream &operator>>(std::istream &is, std::array<S, _siz> &arr) { for (auto &x: arr) is >> x; return is; }
template<typename S, std::size_t _siz> std::ostream &operator<<(std::ostream &os, std::array<S, _siz> arr) { os << "("; for (auto x: arr) os << x << ", "; os << ")"; return os; }
template<typename T> std::istream &operator>>(std::istream &is, std::vector<T> &vec) { for (auto &x: vec) is >> x; return is;}
template<typename T> std::ostream &operator<<(std::ostream &os, const std::vector<T> &vec) { os << '{'; for (auto &x: vec) os << x << ", "; return os << "}";}
#ifdef std_cpp17
template<class Tuple, std::size_t... Is> void print_tuple_impl(std::ostream &os, const Tuple &t, std::index_sequence<Is...>) { ((os << (Is == 0? "" : ", ") << std::get<Is>(t)), ...); }
template<class... Args> std::ostream &operator<<(std::ostream &os, const std::tuple<Args...> &t) { os << "("; print_tuple_impl(os, t, std::index_sequence_for<Args...>{}); return os << ")"; }
#endif
#ifdef DEBUG
#ifdef InTerminal
#define dbg(x...) do { std::cerr << "\033[32;1m" << #x << " -> "; err(x); } while (0)
void err() { std::cerr << "\033[39;0m" << std::endl; }
#else
#define dbg(x...) do { std::cerr  << #x << " -> "; err(x); } while (0)
void err() { std::cerr << std::endl; }
#endif
template<typename T, typename... A>
void err(T a, A... x) { std::cerr << a << ' '; err(x...); }
#else
#define dbg(...)
#endif

using namespace std;
const int maxn = 2e5 + 3;
const int INF = 0x3f3f3f3f3f3f3f3f;
const int mod = 998244353;
mt19937 RD(time(0));


void solv() {


    return ;
}

signed main() {
    // freopen("./data.in", "r", stdin);
    std::ios::sync_with_stdio(false), std::cin.tie(0), std::cout.tie(0);
```

```
    int beg__TT = clock();

    signed _ttt;
    cin >> _ttt;

    while(_ttt--)
        solv();

    #ifdef DEBUG
    std::cerr << "use : " << (clock() - beg__TT) << "ms\n";
    #endif
    return 0;
}
```

数据结构

ST 表

- 一维

```
class Sparcetable {
    vector<vector<int> > st;
    int siz;
    bool MX_flg = 0;
    inline int renew(int x, int y) {
        if (MX_flg) return  max(x, y);
        return min(x, y);
    }
public:
    // 注意 bhpos(0) 返回-1
    bool (*comp)(int, int);
    Sparcetable():siz(maxn) {st.resize(hbpos(maxn - 1) + 1, std::vector<int> (maxn));}
    Sparcetable(const std::vector<int>& a, bool _MX_flg = 1): siz(a.size()), MX_flg(_MX_flg) {
        int n = a.size();
        st.resize(hbpos(n) + 1, vector<int> (n + 1));
        for (int i = 1; i <= n; ++i) st[0][i] = a[i];
        for (int i = 1; i <= hbpos(siz); ++i) {
            for (int j = 1; j + (1 << i) <= siz + 1; ++j) {
                st[i][j] = renew(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
            }
        }
    }
    int query(int l, int r) {
        int len = hbpos(r - l + 1);
        return renew(st[len][l], st[len][r - (1 << len) + 1]);
    }
};

```

- 二维

```
int f[10][10][maxn][maxn];
#define _highbit(x) (31 - __builtin_clz(x))
inline int calc(int x, int y, int xx, int yy, int p, int q) {
    return max(
        max(f[p][q][x][y], f[p][q][xx - (1 << p) + 1][yy - (1 << q) + 1]),
        max(f[p][q][xx - (1 << p) + 1][y], f[p][q][x][yy - (1 << q) + 1])
    );
}
void init() {
    for (int x = 0; x <= _highbit(n); ++x)
    for (int y = 0; y <= _highbit(m); ++y)
        for (int i = 0; i <= n - (1 << x); ++i)
        for (int j = 0; j <= m - (1 << y); ++j) {
            if (!x && !y) { f[x][y][i][j] = a[i][j]; continue; }
            f[x][y][i][j] = calc(
                i, j,
                i + (1 << x) - 1, j + (1 << y) - 1,
                max(x - 1, 0), max(y - 1, 0)
            );
```

```
        }
    }
    inline int get_max(int x, int y, int xx, int yy) {
        return calc(x, y, xx, yy, _highbit(xx - x + 1), _highbit(yy - y + 1));
    }
```

Fenwick Tree(树状数组)

- 一维

```
template<typename T>
class FenwickT {
    int n;
    vector<T> tr;
public:
    FenwickT(int siz): tr(siz), n(siz) {}
    FenwickT(int siz, T ini): tr(siz, ini), n(siz) {}
    void add(int p, T x) {
        for (int i = p; i < n; i += i & (-i)) tr[i] += x;
    }
    T query(int p) {
        T ret = T(0);
        for (int i = p; i > 0; i -= i & (-i)) ret += tr[i];
        return ret;
    }
    T range_sum(int l, int r) {
        return (query(r) - query(l - 1));
    }
    int kth(T k) {
        int x = 0;
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i - 1]) {
                x += i;
                k -= a[x - 1];
            }
        }
        return x;
    }
};
```

并查集

```
class DSU {
    std::vector<int> fa, siz;
public:
    DSU(int n): fa(n), siz(n, 1) {std::iota(fa.begin(), fa.end(), 0);}
    int findfa(int x) {
        int rt = x;
        while (rt != fa[rt]) rt = fa[rt];
        for (int i = x; i != rt; i = x) {
            siz[i] -= siz[x];
            x = fa[i], fa[i] = rt;
        }
        return rt;
    }
    bool merge(int x, int y) {
        x = findfa(x), y = findfa(y);
        if (x == y) return false;
        if (siz[x] > siz[y])
            fa[y] = x, siz[x] += siz[y];
        else
            fa[x] = y, siz[y] += siz[x];
        return true;
    }
    bool same(int x, int y) { return findfa(x) == findfa(y); }
    int size(int x) { return siz[findfa(x)]; }
};
```

带权并查集

```cpp
class DSU {
    std::vector<int> fa, siz;
    std::vector<int> tag;
public:
    DSU(int n): fa(n), siz(n, 1), tag(n, 0) {std::iota(fa.begin(), fa.end(), 0);}
    int findfa(int x) {
        int rt = x, tg = tag[x];
        while (rt != fa[rt]) rt = fa[rt], tg ^= tag[rt];
        for (int i = x; i != rt; i = x) {
            siz[i] -= siz[x];
            x = fa[i], fa[i] = rt;
            // std::swap(tg, tag[i]); tg ^= tag[i];
            tg ^= tag[i] ^= tg ^= tag[i] ^= tg;
        }
        return rt;
    }
    bool merge(int x, int y, int tg = 0) {
        if (findfa(x) == findfa(y)) {
            if ((tag[x] ^ tag[y]) != tg) return false;
            return true;
        }
        tg ^= tag[x] ^ tag[y];
        x = findfa(x), y = findfa(y);
        if (siz[x] > siz[y])
            fa[y] = x, siz[x] += siz[y], tag[y] = tg;
        else
            fa[x] = y, siz[y] += siz[x], tag[x] = tg;
        return true;
    }
    bool same(int x, int y) { return findfa(x) == findfa(y); }
    int size(int x) { return siz[findfa(x)]; }
};

```

线段树

```cpp
class SegT{
    struct Pt{
        int val, ls, rs, sl;
        Pt():val(0), ls(0), rs(0), sl(0) {}
        Pt(int v, int l, int r, int s): val(v), ls(l), rs(r), sl(s) {}
        ~Pt() {}
    };
    int renew(int x, int y) {
        return x + y;
    }
    int renewsl(int x, int y) {
        return x + y;
    }
    int renewqj(int x, int y, int l, int r) {
        return x + y * (r - l + 1);
    }
public:
    std::vector<Pt> tr;
    int cnt = 0; // 结点数目
    SegT() {cnt = 1; tr.emplace_back();} // 建一棵空树

    // 修改
    void add(int l, int r, int p, int ql, int qr, int x) {
        if (ql <= l && r <= qr) {
            tr[p].val = renewqj(tr[p].val, x, l, r);
            tr[p].sl = renewsl(tr[p].sl, x);
            return;
        }
        int mid = l + (r - l) / 2;
        if (tr[p].sl) {
            if (!tr[p].ls) tr[p].ls = cnt++;
            if (!tr[p].rs) tr[p].rs = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
```

```
            int ls = tr[p].ls, rs = tr[p].rs;
           tr[ls].val = renewqj(tr[ls].val, tr[p].sl, l, mid), tr[rs].val = renewqj(tr[rs].val, tr[p].sl, mid + 1, r);
            tr[ls].sl = renewsl(tr[ls].sl, tr[p].sl), tr[rs].sl = renewsl(tr[rs].sl, tr[p].sl);
            tr[p].sl = 0;
        }
        if (ql <= mid) {
            if (!tr[p].ls) tr[p].ls = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            add(l, mid, tr[p].ls, ql, qr, x);
        }
        if (qr > mid) {
            if (!tr[p].rs) tr[p].rs = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            add(mid + 1, r, tr[p].rs, ql, qr, x);
        }
        tr[p].val = renew(tr[p].ls? tr[tr[p].ls].val: 0, tr[p].rs? tr[tr[p].rs].val: 0);
    };

    // 查询
    int query(int l, int r, int p, int ql, int qr) {
        if (ql <= l && r <= qr) return tr[p].val;
        int mid = l + (r - l) / 2;
        if (tr[p].sl) {
            if (!tr[p].ls) tr[p].ls = cnt++;
            if (!tr[p].rs) tr[p].rs = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            int ls = tr[p].ls, rs = tr[p].rs;
           tr[ls].val = renewqj(tr[ls].val, tr[p].sl, l, mid), tr[rs].val = renewqj(tr[rs].val, tr[p].sl, mid + 1, r);
            tr[ls].sl = renewsl(tr[ls].sl, tr[p].sl), tr[rs].sl = renewsl(tr[rs].sl, tr[p].sl);
            tr[p].sl = 0;
        }
        int ret = 0;
        if (ql <= mid) {
            if (!tr[p].ls) tr[p].ls = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            ret = renew(ret, query(l, mid, tr[p].ls, ql, qr));
        }
        if (qr > mid) {
            if (!tr[p].rs) tr[p].rs = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            ret = renew(ret, query(mid + 1, r, tr[p].rs, ql, qr));
        }
        tr[p].val = renew(tr[p].ls? tr[tr[p].ls].val: 0, tr[p].rs? tr[tr[p].rs].val: 0);
        return ret;
    };
};

// 静态线段树
template<class Info,
    class Merge = std::plus<Info> >
struct SegmentTree {
    const int n;
    const Merge merge;
    std::vector<Info> info;
    SegmentTree(int n) : n(n), merge(Merge()), info(4 << std::__lg(n)) {}
    SegmentTree(std::vector<Info> init) : SegmentTree(init.size()) {
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = merge(info[2 * p], info[2 * p + 1]);
    }
```

```
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
};
```

zkw-SegmentTree

```
template<typename E>
class zkwSegT {
    std::vector<E> tr;
    int _n, N;
public:
    zkwSegT(int siz_ = 0) {
        _n = siz_;
        N = 1;
        while (N < _n) N <<= 1;
        tr.resize(N << 1);
    }
    zkwSegT(const std::vector<E>& a) {
        _n = a.size();
        N = 1;
        while (N < _n) N <<= 1;
        tr.resize(N << 1);
        for (int i = 0; i < _n; ++i) tr[i + N] = a[i];
        for (int i = N - 1; i >= 1; --i) {
            tr[i] = tr[i << 1] + tr[i << 1 | 1];
        }
    }
    void update(int x, E v) {
        tr[x + N] = v;
        for (int i = (x + N) >> 1; i >= 1; i >>= 1) {
            tr[i] = tr[i << 1] + tr[i << 1 | 1];
        }
    }
    E query(int l, int r) {
        E ans;
        for (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
            if (~l & 1) ans = ans + tr[l ^ 1];
            if ( r & 1) ans = ans + tr[r ^ 1];
        }
        return ans;
    }
};


```

李超线段树

```cpp
// luogu 1e5: 60ms
using pdi = std::pair<double,int>;
constexpr double eps = 1e-8;
struct Line {
    double k, b;
    int id;
    Line() : k(0),b(-INF),id(-1) {}
    Line(int x0,int y0,int x1,int y1,int id) : id(id) {
        if(x0 == x1) {
            k = 0, b = std::max(y0, y1);
            return;
        }else if(x0 > x1) {
            std::swap(x0,x1), std::swap(y0,y1);
        }
        k = 1.0 * (y1 - y0) / (x1 - x0);
        b = y0 - x0 * k;
    }
    double cal(int x) {
        return k * x + b;
    }
};
int sgn(double x) {
    if(fabs(x) <= eps) return 0;
    else if(x > 0) return 1;
    else return -1;
}
pdi merge(pdi x,pdi y) {
    int s = sgn(x.first - y.first);
    if(s > 0) return x;
    else if(s < 0) return y;
    else return pdi(x.first, std::min(x.second, y.second));
}
struct LiChaoTree {
    int n;
    std::vector<Line> t;
    LiChaoTree(int n) : n(n),t(4*n) {}
    void modify(int p,int l,int r,int L,int R,Line v) {
        // [L,R)
        int mid = l + r >> 1;
        if(R <= l || r <= L) {
            return;
        }
        if(L <= l && r <= R) {
            Line &u = t[p];
            if(sgn(u.cal(mid) - v.cal(mid)) < 0) std::swap(u,v); // u.mid >= v.mid
            if(sgn(u.cal(l) - v.cal(l)) < 0) modify(p<<1,l,mid,L,R,v);
            if(sgn(u.cal(r-1) - v.cal(r-1)) < 0) modify(p<<1|1,mid,r,L,R,v);
            return;
        }
        modify(p<<1,l,mid,L,R,v);
        modify(p<<1|1,mid,r,L,R,v);
    }
    void modify(int L,int R,Line v) {
        modify(1,0,n,L,R,v);
    }
    pdi query(int p,int l,int r,int x) {
        int mid = l + r >> 1;
        double thiz = t[p].cal(x);
        if(r - l == 1) return pdi(thiz, t[p].id);
        else if(x < mid) return merge(pdi(thiz, t[p].id), query(p<<1,l,mid,x));
        else return merge(pdi(thiz, t[p].id), query(p<<1|1,mid,r,x));

    }
    pdi query(int x) {
        return query(1,0,n,x);
    }
};

```

吉如一线段树

区间赋值、区间取 min，求和、求最大值、求历史最大值

```cpp
namespace JRY_SegT{
    // namespace JRY_SegT
    // O(n log n) luogu n,m 5e5 max1.62s
    using i64 = long long;
    constexpr i64 INF = 0x7fffffffffffffff;
    struct Info {
        i64 sum,mx,smx,hmx,cnt;
        Info(i64 sum=0,i64 mx=-INF,i64 smx=-INF,i64 hmx=-INF,i64 cnt=0)
            : sum(sum),mx(mx),smx(smx),hmx(hmx),cnt(cnt) {}
    };
    Info merge(Info x,Info y) {
        Info r;
        r.sum = x.sum + y.sum;
        r.mx = std::max(x.mx, y.mx);
        r.hmx = std::max(x.hmx, y.hmx);
        if(x.mx == y.mx) r.smx = std::max(x.smx, y.smx), r.cnt = x.cnt + y.cnt;
        else if(x.mx > y.mx) r.smx = std::max(x.smx, y.mx), r.cnt = x.cnt;
        else r.smx = std::max(x.mx, y.smx), r.cnt = y.cnt;
        return r;
    }
    struct Tag {
        i64 lzm,lzo,hlzm,hlzo;
        Tag() : lzm(0),lzo(0),hlzm(0),hlzo(0) {}
        void clear() {
            lzm = lzo = hlzm = hlzo = 0;
        }
        bool is0() {
            return lzm == 0 && lzo == 0 && hlzm == 0 && hlzo == 0;
        }
    };
    struct SegT {
        int n;
        std::vector<Info> t;
        std::vector<Tag> z;
        std::vector<int> le,ri;
        SegT(int n,const std::vector<i64>&a) : n(n), t(4*n), z(4*n), le(4*n), ri(4*n) {
            build(1,0,n,a);
        }
        void build(int p,int l,int r,const std::vector<i64>&a) {
            le[p] = l, ri[p] = r;
            if(r - l == 1) {
                t[p] = Info(a[l],a[l],-INF,a[l],1);
                return;
            }
            int mid = l + r >> 1;
            build(p << 1, l, mid, a);
            build(p << 1 | 1, mid, r, a);
            pull(p);
        }
        void pull(int p) {
            t[p] = merge(t[p << 1], t[p << 1 | 1]);
        }
        void update(int p,i64 lzm,i64 lzo,i64 hlzm,i64 hlzo) {
            t[p].sum += lzm * t[p].cnt + (ri[p] - le[p] - t[p].cnt) * lzo;
            t[p].hmx = std::max(t[p].hmx, t[p].mx + hlzm);
            t[p].mx += lzm;
            if(t[p].smx != -INF) t[p].smx += lzo;
            z[p].hlzm = std::max(z[p].hlzm, z[p].lzm + hlzm), z[p].lzm += lzm;
            z[p].hlzo = std::max(z[p].hlzo, z[p].lzo + hlzo), z[p].lzo += lzo;
        }
        void pushdown(int p) {
            if(z[p].is0()) return;
            int mx = std::max(t[p << 1].mx, t[p << 1 | 1].mx);
            for(auto ps : {p << 1, p << 1 | 1}) {
                if(t[ps].mx == mx) update(ps, z[p].lzm, z[p].lzo, z[p].hlzm, z[p].hlzo);
                else update(ps, z[p].lzo, z[p].lzo, z[p].hlzo, z[p].hlzo);
            }
            z[p].clear();
```

```
        }
        void range_add(int p,int L,int R,i64 v) {
            if(ri[p] <= L || R <= le[p]) return;
            if(L <= le[p] && ri[p] <= R) {
                update(p,v,v,v,v);
                return;
            }
            pushdown(p);
            range_add(p << 1, L, R, v);
            range_add(p << 1 | 1, L, R, v);
            pull(p);
        }
        void range_min(int p,int L,int R,i64 v) {
            if(ri[p] <= L || R <= le[p] || t[p].mx <= v) return;
            if(L <= le[p] && ri[p] <= R && t[p].smx < v) {
                update(p, v - t[p].mx, 0, v - t[p].mx, 0);
                return;
            }
            pushdown(p);
            range_min(p << 1, L, R, v);
            range_min(p << 1 | 1, L, R, v);
            pull(p);
        }
        i64 query_sum(int p,int L,int R) {
            if(ri[p] <= L || R <= le[p]) return 0ll;
            if(L <= le[p] && ri[p] <= R) return t[p].sum;
            pushdown(p);
            return query_sum(p << 1, L, R) + query_sum(p << 1 | 1, L, R);
        }
        i64 query_mx(int p,int L,int R) {
            if(ri[p] <= L || R <= le[p]) return -INF;
            if(L <= le[p] && ri[p] <= R) return t[p].mx;
            pushdown(p);
            return std::max(query_mx(p << 1, L, R), query_mx(p << 1 | 1, L, R));
        }
        i64 query_hmx(int p,int L,int R) {
            if(ri[p] <= L || R <= le[p]) return -INF;
            if(L <= le[p] && ri[p] <= R) return t[p].hmx;
            pushdown(p);
            return std::max(query_hmx(p << 1, L, R), query_hmx(p << 1 | 1, L, R));
        }
    };
};

```

主席树

可持久化线段树, 感觉主要是利用单点 log 次的性质以及离散化的思路. 比如区间第 k 小就是构造 n 个线段树 (每多加一个实际上只多加了 log 个点), 然后再用离散化的点 (实际上就是其排名值) 找到前缀和之差为 k(此时为第 k 小, 可以这样想: 第 r 棵树的 r 个点中包含了第 l-1 棵树的 l-1 个点, 减掉那些点, 剩下的就是区间 [l, r] 的点, 然后找第 k 个) 的点. 用离线不那么方便. 代码暂时先留空了...

```
// 主席树
// 对权值建立可持久化线段树，再二分一样求区间第 k 大
struct HJTtr {
    std::vector<int> ls, rs, rt;
    std::vector<int> ind, sum;
    int tot, _n;
    int getid(const int& val) {
        return std::lower_bound(ind.begin(), ind.end(), val) - ind.begin();
    }
    int build(int l, int r) {
        int root = tot++;
        sum[root] = 0;
        if (l == r) return root;
        int mid = l + (r - l) / 2;
        ls[root] = build(l, mid);
        rs[root] = build(mid + 1, r);
        return root;
    }
```

```
    int update(int k, int x = 1) {
        auto upd=[&](auto upd, int l, int r, int root) {
            int dir = tot++;
            ls[dir] = ls[root], rs[dir] = rs[root], sum[dir] = sum[root] + x;
            if (l == r) return dir;
            int mid = l + (r - l) / 2;
            if (k <= mid) ls[dir] = upd(upd, l, mid, ls[dir]);
            else rs[dir] = upd(upd, mid + 1, r, rs[dir]);
            return dir;
        };
        return upd(upd, 0, _n - 1, rt.back());
    }
    HJTtr(int n): ls(n * 20), rs(n * 20), ind(n), sum(n * 20), tot(0), _n(n) {}
    void init(const std::vector<int> &a) {
        tot = 0;
        ind.assign(a.begin(), a.end());
        std::sort(ind.begin(), ind.end());
        ind.erase(std::unique(ind.begin(), ind.end()), ind.end());
        rt.push_back(build(0, _n - 1));
        for (int i = 0; i < a.size(); ++i) rt.push_back(update(getid(a[i])));
    }
    /// @brief  查询区间第 k 大，注意区间是左闭右闭。
    /// @param l 左区间
    /// @param r 右区间
    /// @param k 第 k 大
    /// @return
    /// @note   l, r, k are all 1-indexed.
    int query(int l, int r, int k) {
        auto qry=[&](auto qry, int u, int v, int vl, int vr, int kk) {
            if (vl == vr) return ind[vl];
            int mid = vl + (vr - vl) / 2, cnt = sum[ls[v]] - sum[ls[u]];
            if (kk <= cnt) return qry(qry, ls[u], ls[v], vl, mid, kk);
            else return qry(qry, rs[u], rs[v], mid + 1, vr, kk - cnt);
        };
        return qry(qry, rt[l-1], rt[r], 0, _n - 1, k);
    }
};


```

Trie

```
template<typename T>
struct Trie {
    struct Node {
        T val;
        int to[2];
        Node(): val{0} {memset(to, 0, sizeof(to));}
        Node(T v): val(v) {memset(to, 0, sizeof(to));}
    };
    std::vector<Node> a;
    int cnt;
    Trie(): a(1), cnt(1) {}
    Trie(int siz): a(siz), cnt(1) {}
    int at(char ch) {return ch - '0';}
    Node& operator [](int idx) {return a[idx];}
    int go(int cur, char ch) {
        int &stat = a[cur].to[at(ch)];
        if (stat == 0) {
            stat = cnt++;
            while (cnt > a.size()) a.emplace_back();
        }
        return a[cur].to[at(ch)];
    }
};

```

笛卡尔树

本质就是一个单调栈，可以用来求一个序列的最大子序列的左右端点

```cpp
std::vector<std::array<int, 2> > g;
int dikaer(std::vector<int> a) {
    std::stack<int> stk;
    g.resize(a.size());
    for (int i = 1; i < a.size(); ++i) {
        int las = -1;
        while (!stk.empty() && a[stk.top()] <= a[i]) las = stk.top(), stk.pop();
        if (~las) g[i][0] = las;
        if (!stk.empty()) g[stk.top()][1] = i;
        stk.push(i);
    }
    int rt = -1;
    while (!stk.empty()) rt = stk.top(), stk.pop();
    return rt;
}

```

treap

fhq-treap

```cpp
// Definition
class Treap {
  private:
    struct node {
        node *left, *right;
        int size, val, key;

        node();
        node(int);
        ~node();

        void pushup();
    } * root;

    int getNodeSize(node *);
    node *find(node *, int);
    std::pair<node *, node *> split(node *, int);
    std::pair<node *, node *> splitByValue(node *, int);
    node *merge(node *, node *);

  public:
    Treap();
    ~Treap();

    void insert(int);
    void erase(int);
    int getRank(int);
    int getKth(int);
    void print(node *p, int dep);
    void print() { print(root, 0); }
} ;

// === Treap ===

// struct Treap::node

std::mt19937 Rnd(std::chrono::steady_clock::now().time_since_epoch().count());

Treap::node::node()
    : left(nullptr), right(nullptr), size(0), val(0), key(Rnd()) {}

Treap::node::node(int _val)
    : left(nullptr), right(nullptr), size(1), val(_val), key(Rnd()) {}

Treap::node::~node() {
    delete left, right;
}

inline void Treap::node::pushup() {
    size = 1;
```

```
51      if (left != nullptr) size += left->size;
52      if (right != nullptr) size += right->size;
53  }
54
55  // class Treap
56
57  Treap::Treap()
58      : root(nullptr) {}
59
60  Treap::~Treap() {
61      delete root;
62  }
63
64  inline int Treap::getNodeSize(Treap::node *node) {
65      return node == nullptr ? 0 : node->size;
66  }
67
68  std::pair<Treap::node *, Treap::node *> Treap::split(Treap::node *p, int k) {
69      if (p == nullptr) return std::make_pair(nullptr, nullptr);
70      std::pair<Treap::node *, Treap::node *> o;
71      if (k <= getNodeSize(p->left)) {
72          o = split(p->left, k);
73          p->left = o.second;
74          p->pushup();
75          o.second = p;
76      } else {
77          o = split(p->right, k - getNodeSize(p->left) - 1);
78          p->right = o.first;
79          p->pushup();
80          o.first = p;
81      }
82      return o;
83  }
84
85  std::pair<Treap::node *, Treap::node *> Treap::splitByValue(Treap::node *p, int val) {
86      if (p == nullptr) return std::make_pair(nullptr, nullptr);
87      std::pair<Treap::node *, Treap::node *> o;
88      if (p->val < val) {
89          o = splitByValue(p->right, val);
90          p->right = o.first;
91          p->pushup();
92          o.first = p;
93      } else {
94          o = splitByValue(p->left, val);
95          p->left = o.second;
96          p->pushup();
97          o.second = p;
98      }
99      return o;
100 }
101
102 Treap::node *Treap::merge(Treap::node *x, Treap::node *y) {
103     if (x == nullptr) return y;
104     if (y == nullptr) return x;
105     if (x->key > y->key) {
106         x->right = merge(x->right, y);
107         x->pushup();
108         return x;
109     }
110     y->left = merge(x, y->left);
111     y->pushup();
112     return y;
113 }
114
115 Treap::node *Treap::find(Treap::node *p, int val) {
116     if (p == nullptr) return nullptr;
117     if (p->val == val) return p;
118     if (p->val > val) return find(p->left, val);
119     return find(p->right, val);
120 }
121
```

```cpp
void Treap::insert(int val) {
    auto o = splitByValue(root, val);
    o.first = merge(o.first, new Treap::node(val));
    root = merge(o.first, o.second);
}

void Treap::erase(int val) {
    auto o = splitByValue(root, val);
    auto t = o;
    if (find(o.second, val) != nullptr) {
        t = split(o.second, 1);
        delete t.first;
    }
    root = merge(o.first, t.second);
}

int Treap::getRank(int val) {
    auto x = splitByValue(root, val);
    int r = getNodeSize(x.first) + 1;
    root = merge(x.first, x.second);
    return r;
}

int Treap::getKth(int k) {
    auto x = split(root, k - 1);
    auto y = split(x.second, 1);
    Treap::node *o = y.first;
    root = merge(x.first, merge(y.first, y.second));
    return o == nullptr ? 0 : o->val;
}

void Treap::print(Treap::node *p, int dep) {
    if (p == nullptr) return;
    print(p->left, dep + 1);
    for (int i = 0; i < dep; ++i) std::cout << "  ";
    std::cout << p->val << '\n';
    print(p->right, dep + 1);
}

```

pbds_treap

```cpp
// #include<bits/extc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
/*
// tag: rb_tree_tag, splay_tree_tag, ov_tree_tag
// template<Key, Mapped, Cmp_Fn, Tag, Template<Node_CItr, Node_Itr, Cmp_Fn, _Alloc> class Node_Update, _Alloc>
// tree::find(val) / tree::insert(val) / tree::erase(val)
// tree::join(tree) / tree::split(key, RBtree) (将大于 key 的元素放到 RBtree 中) /
// tree::lower_bound(val) / tree::upper_bound(val)
// tree::find_by_order(k) / tree::order_of_key(k)
 */
using Tree = __gnu_pbds::tree<std::array<int,2>,
                __gnu_pbds::null_type,
                std::less<std::array<int,2>>,
                __gnu_pbds::rb_tree_tag,
                __gnu_pbds::tree_order_statistics_node_update>;

```

数学

数论

数论整数

```cpp
constexpr int P = 998244353;
// assume -P <= x < 2P
int norm(int x) {
```

```
    // x %= P;
    if (x < 0) { x += P; }
    if (x >= P) { x -= P; }
    return x;
}
template<typename E>
E power(E n, int k) {
    E ret = E(1);
    while (k) {
        if (k & 1) ret *= n;
        n *= n;
        k >>= 1;
    } return ret;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    int val() const { return x; }
    Z operator-() const { return Z(norm(P - x)); }
    Z inv() const { assert(x != 0); return power(*this, P - 2); }
    Z &operator*=(const Z &rhs) { x = (long long)(x) * rhs.x % P; return *this; }
    Z &operator+=(const Z &rhs) { x = norm(x + rhs.x); return *this; }
    Z &operator-=(const Z &rhs) { x = norm(x - rhs.x); return *this; }
    Z &operator/=(const Z &rhs) { return *this *= rhs.inv(); }
    friend Z operator*(const Z &lhs, const Z &rhs) { Z res = lhs; res *= rhs; return res; }
    friend Z operator+(const Z &lhs, const Z &rhs) { Z res = lhs; res += rhs; return res; }
    friend Z operator-(const Z &lhs, const Z &rhs) { Z res = lhs; res -= rhs; return res; }
    friend Z operator/(const Z &lhs, const Z &rhs) { Z res = lhs; res /= rhs; return res; }
    friend std::istream &operator>>(std::istream &is, Z &a) { long long v; is >> v; a = Z(v); return is; }
    friend std::ostream &operator<<(std::ostream &os, const Z &a) { return os << a.val(); }
};
Z operator"" _z(unsigned long long x) { return Z(x); }
```

拉格朗日差值法

```
template<typename T>
T lintp(const vector<int>& x, const vector<T>& y, int k) {
    T ans = 0;
    for (int i = 0; i < x.size(); ++i) {
        if (k == x[i])
            return y[i];
        T u = 1, v = 1;
        for (int j = 0; j < x.size(); ++j) {
            if (i == j) continue;
            u *= (k - x[j]);
            v *= (x[i] - x[j]);
        }
        ans += y[i] * u / v;
    }
    return ans;
}
```

欧几里得

- 扩展欧几里得

```
int exgcd(int a, int b, int& x, int& y) {
    if (a == 0) {
        x = 0, y = 1;
        return b;
    }
    int ret = exgcd(b % a, a, x, y), xx = x;
    x = y - b / a * x;
    y = xx;
    return ret;
}
```

类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.

- $f(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor$：当 $a\geq c$ or $b\geq c$ 时，$f(a,b,c,n)=(\frac{a}{c})n(n+1)/2+(\frac{b}{c})(n+1)+f(a \bmod c,b \bmod c,c,n)$；否则 $f(a,b,c,n)=nm-f(c,c-b-1,a,m-1)$。
- $g(a,b,c,n)=\sum_{i=0}^{n}i\lfloor\frac{ai+b}{c}\rfloor$：当 $a\geq c$ or $b\geq c$ 时，$g(a,b,c,n)=(\frac{a}{c})n(n+1)(2n+1)/6+(\frac{b}{c})n(n+1)/2+g(a \bmod c,b \bmod c,c,n)$；否则 $g(a,b,c,n)=\frac{1}{2}(n(n+1)m-f(c,c-b-1,a,m-1)-h(c,c-b-1,a,m-1))$。
- $h(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor^2$：当 $a\geq c$ or $b\geq c$ 时，$h(a,b,c,n)=(\frac{a}{c})^2n(n+1)(2n+1)/6+(\frac{b}{c})^2(n+1)+(\frac{a}{c})(\frac{b}{c})n(n+1)+h(a \bmod c,b \bmod c,c,n)+2(\frac{a}{c})g(a \bmod c,b \bmod c,c,n)+2(\frac{b}{c})f(a \bmod c,b \bmod c,c,n)$；否则 $h(a,b,c,n)=nm(m+1)-2g(c,c-b-1,a,m-1)-2f(c,c-b-1,a,m-1)-f(a,b,c,n)$。

```cpp
template<typename T>
T lintp(const vector<int>& x, const vector<T>& y, int k) {
    T ans = 0;
    for (int i = 0; i < x.size(); ++i) {
        if (k == x[i])
            return y[i];
        T u = 1, v = 1;
        for (int j = 0; j < x.size(); ++j) {
            if (i == j) continue;
            u *= (k - x[j]);
            v *= (x[i] - x[j]);
        }
        ans += y[i] * u / v;
    }
    return ans;
}
```

Miller-Rabin 和 Pollard-Rho

- Miller-Rabin

```cpp
long long pow_128(__int128_t n, long long k, long long mo) {
    __int128_t ret = 1;
    while (k) {
        if (k & 1) (ret *= n) %= mo;
        (n *= n) %= mo;
        k >>= 1;
    } return ret;
}

bool miller_rabin(long long n) {
    static const long long jp[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };

    if (n <= 1 || (n > 2 && !(n & 1))) return false;
    for (long long p : jp) if (n % p == 0) return n == p;

    long long r = n - 1, x, y;
    long long e = 0;
    while (~r & 1) r >>= 1, ++e;
    for (long long p : jp) {
        x = pow_128(p, r, n);
        for (long long t = 0; t < e && x > 1; ++t) {
            y = (__int128_t)x * x % n;
            if (y == 1 && x != n - 1) return false;
            x = y;
        }
        if (x != 1) return false;
    }
    return true;
}

```

- Pollar-rho

```cpp
std::mt19937 RD(time(0));
long long pollard_rho(long long x) {
    long long s = 0, t = 0;
    long long c = (long long)RD() % (x - 1) + 1;
    int step = 0, goal = 1;
    long long val = 1;
    for (goal = 1;; goal *= 2, s = t, val = 1) { // 倍增优化
        for (step = 1; step <= goal; ++step) {
```